Optimistic Heuristics for MineSweeper

Olivier Buffet[3], Chang-Shing Lee[2], Woan-Tyng Lin[2], Olivier Teytaud[1,2,4]

[1] TAO-INRIA, LRI, CNRS UMR 8623,
Université Paris-Sud, Orsay, France
[2] OASE Lab, National University of Tainan, Taiwan
[3] MAIA team, LORIA, INRIA/Universit de Lorraine, France
[4] Montefiore Institute, Université de Liège, Belgium

Abstract. We present a combination of Upper Confidence Tree (UCT) and domain specific solvers, aimed at improving the behavior of UCT for long term aspects of a problem. Results improve the state of the art, combining top performance on small boards (where UCT is the state of the art) and on big boards (where variants of CSP rule).

1 Introduction

This sequential decision making research is based on the following requirements on our favorite application:

- anytime algorithm (behavior for unknown and possibly small time settings): we want our algorithm to be able to run in very limited time and provide an approximate solution. For example, Direct Policy Search (DPS) and Upper Confidence Tree (UCT) satisfy this requirement.
- asymptotic optimality (behavior for long time settings): many algorithms are fast, but have strong assumptions on the model, so that the real problem is modified in order to match the model requirement; as a consequence, the solution, even if it is optimal on the model, is not optimal on the real problem. For example, UCT satisfies this requirement (in discrete cases, and with some modifications also in continuous domains); many reinforcement learning tools also have this property. DPS, on the other hand, is usually limited by the policy structure.
- compatibility with long term effects: many Markov Decision Processes (MDP) have long term effects; you can not make optimal decisions unless you take into account long term consequences of your actions. This requirement is harder to formalize: Direct Policy Search has no problem with this, because it is based on complete simulations of the problem; also, Stochastic Dynamic Programming takes into account long term effects. On the other hand, it is not the case for UCT. Theoretically, UCT will asymptotically find optimal solutions if the horizon is finite; but in real life, only the few initial time steps make sense, the remaining part being left to the pure Monte-Carlo part.

– compatibility with high-dimension: many works are dedicated to improving the scalability of MDP-solving algorithms; a great success for this is UCT, with its ability to focus on the relevant parts of the search space.

In this paper, we propose the following principle:

– Using a UCT algorithm for handling the short term combinatorial explosion of the tree of possible futures.
– Using, in lieu of the standard pure Monte-Carlo simulator of UCT, a solver known for good performance for long-term behavior.

Section 2 presents some tools useful in the paper. Section 3 presents our algorithm, Optimistic Heuristics. Section 4 presents experimental results. Section 5 concludes.

2 Tools

In this section, we define the tools on which our algorithm is based: Upper Confidence Trees (Section 2.1), Constraint Satisfaction Problems (Section 2.2).

2.1 Upper Confidence Trees

Upper Confidence Trees[9] proceeds as explained in Alg. 1 for making a decision in a state s in time t.

Algorithm 1 The UCT algorithm in short.

Inputs: a state s, a time t.
Output: a decision.
while t is not elapsed **do**
  $s' = s$ // initialization of a simulation
  while s' is not a final state // this loop is a simulation **do**
    if s' has more than 5 simulations **then**
      Let d be the decision in s' which maximizes $Q_{ucb}(s', d)$
    else
      Let d be a decision in s' chosen by a heuristic (a.k.a Monte-Carlo part)
    end if
    Let s'' be a (possibly stochastic) next state obtained after decision d in s'.
    $s' \leftarrow s''$
  end while
  Update Q_{ucb} values and $\hat{Q}$ values
end while
Return the decision in s' with maximum $\hat{Q}(s', d)$.

This algorithm uses the two following formulas:

$$\hat{Q}(s,d) = \text{ mean reward of past simulations including decision } d \text{ in state } s$$

$$Q_{ucb}(s,d) = \hat{Q}(s,d) + \sqrt{\frac{\log(2+n(s))}{1+n(s,d)}}$$

where $n(s,d)$ is the number of simulations with move d in state s and $n(s) = \sum_d n(s,d)$.

It is widely reported that choosing a heuristic with high performance does not necessarily lead to high performance of the UCT built on top of it [5]. However, in the one player case, introducing strong heuristics is seemingly quite efficient[3, 13]. This is the principle of [13], using a strong heuristic, namely CSP, for improving UCT performance on MineSweeper on small boards.

2.2 Constraint Satisfaction Problems + Heuristics

In some problems in which rigorous optimality is unreachable, just optimizing the instantaneous reward might be a good idea. For example, in the MineSweeper problem, choosing the move with lowest probability of immediate loss is efficient. Nonetheless, it is not sufficient for rigorous optimality; as shown by [11], when two locations have the same probability of mine, choosing the one which is as close as possible to the frontier between covered locations and uncovered locations might be a good idea. For examples, if 6 locations *a, b, c, d, e, f* are uncovered, with probability of mine respectively 50%, 50%, 20%, 20%, 20%, 20%, and distance to the frontier respectively 1, 1, 2, 2, 2, 1, then move (f) should be chosen (at least according to this heuristic), because (i) moves (c), (d), (e) and (f) have lowest probability of mines, and among these locations (f) has smallest distance to frontier. Formally, the frontier is defined as the set of locations which are uncovered and next to at least one mine. This is not formally proved as optimal (and Fig. 2 shows that it is not optimal), but it is on average quite efficient (see results in [11]) as a heuristic for breaking ties between equivalent moves. This heuristic "tie breaking" provides significant improvements. However, as shown by [3], there are cases where the optimal move is difficult to find among moves with minimum probability of mine. Therefore, CSP variants do not have the asymptotic optimality property discussed in the introduction.

3 Optimistic Heuristics

We define Optimistic Heuristics (OH), our UCT-based algorithm, as follows:

- the main structure of the program is UCT;
- we fix the first move at a corner of the board;
- we expand only children with minimum probability of mine;
- but the Monte-Carlo part is replaced by HCSP (Heuristic CSP), a variant of CSP with improved performance thanks to the heuristic of prefering locations close to the frontier between covered and uncovered locations.

Algorithm 2 Our Optimistic Heuristics algorithm. The two parts in bold involve specialized methods, namely CSP and HCSP.

Inputs: a state s, a time t.
Output: a decision.
while t is not elapsed **do**
  $s' = s$ // initialization of a simulation
  while s' is not a final state // this loop is a simulation **do**
    CSP: Let E be the set of moves with minimum probability of mine.
    if s' has more than 5 simulations **then**
      Let d be the decision in E which maximizes $Q_{ucb}(s', d)$
    else
      Let d be a decision in E chosen by HCSP.
    end if
    Let s'' be a (possibly stochastic) next state obtained after decision d in s'.
    $s' \leftarrow s''$
  end while
  Update Q_{ucb} values and $\hat{Q}$ values
end while
Return the decision in s' with maximum $\hat{Q}(s', d)$.

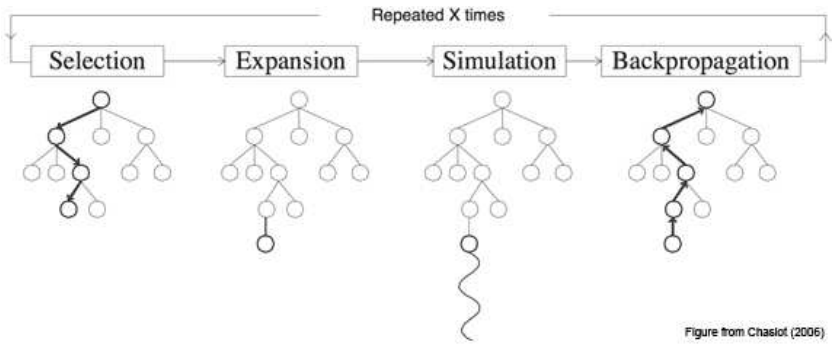

Fig. 1. A description of MCTS in one Figure, from [2]. Before making a decision, the algorithm performs many simulations. Each simulation is typically made of (i) a path in the current tree stored in memory (ii) a construction of one new node added to the tree (iii) a Monte-Carlo simulation from this new node.

The pseudo-code is given in Alg. 2. Describing UCT in details is beyond the scope of this paper; so we refer to `http://www.mcts.ai/?q=mcts` for a user-friendly introduction. Figure 1 provides a graphical overview.

4 Experiments

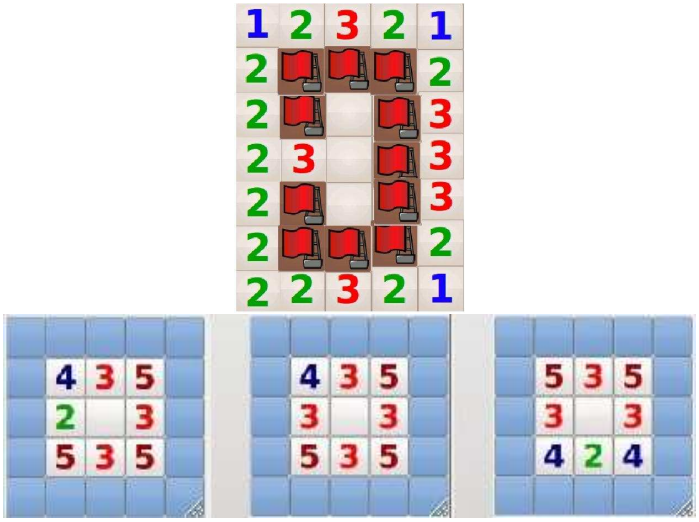

Fig. 2. Situations in which some moves with minimum probability of mine are sub-optimal. We are not aware of situations where no move with minimum probability of mine is optimal; so in our program we expand only the parts of the tree corresponding to moves with lowest probability of mine (yet, we have no proof of optimality for this). Top: the top location and the bottom covered locations lead to a probability of winning $\frac{2}{3}$, whereas the middle location leads to a probability of winning $\frac{1}{3}$. Bottom: these three boards are the three possible cases when playing 5x5 GnoMine with 15 mines (up to rotation): they show that under GnoMine rule, playing in the center (for the first move) leads to a 100% winning rate. One can easily see that the success rate when playing any other move is $< 100\%$. So, this case shows that even for the first move, choosing a move with minimal probability of mine is not enough for playing optimally; moreover, corners are not optimal for the GnoMine variant.

The well known MineSweeper game is much more complex than expected at first sight [8, 1, 12], and a good tool for modeling in a clean way some real world problems[7]. The most classical approach for MineSweeper is based on Constraint Satisfaction Problems[14], as in Section 2.2: this provides a provably correct estimate of the belief state, and then classically after CSP one plays the covered location with least probability of being a mine. However, [3] has shown that this approach is suboptimal; there are situations, as in Fig. 2, where some

moves with minimum probability of mines are nonetheless suboptimal moves. This fact was used in [11] for designing a version of CSP with better performance, by preferring the locations which are closest to the frontier in case of tie on the probability of mines; [11] got the best performance so far on MineSweeper on many board sizes, under time constraints, in particular big boards.

An alternate approach has been investigated in [3, 13]; using UCT, these approaches have asymptotic optimality, and so outperform CSP when given enough time. They got best performance so far on small boards, outperforming CSP, but are too slow for big boards.

[13] and [11] outperformed the state of the art in two different settings; close to optimality on small boards for [13] (compliant with any variant of rule), and better than CSP on big boards for [11]; in this paper we combine both approaches, using UCT with the HCSP approach as a heuristic.

MineSweeper is parametrized by the number of mines, the number of rows, the number of columns. Importantly, there are different variants of MineSweeper:

- variants in which the mines can be anywhere on the board;
- variants in which the mines can not be on the first move (as in most implementations);
- variants in which there are no mine in the neighborhood of the first move (e.g. the GnoMine implementation).

The optimal strategies are the same for the first two variants. It is widely believed that moves in the corners are optimal in the two first cases; but, as shown by the example in Fig. 2 (right) this is not the case for the third family of variants. For the sake of sound comparisons with earlier results, we will present results in the second case, i.e. one can not lose at the first move.

Results are presented in Table 1.

5 Conclusions

We experimented a combination between UCT and CSP solvers. We got a reasonably fast solver, much faster than the one from [13], and benefiting from the heuristic from [11]. Results are at the state of the art on all board sizes.

Further work consists in

- Pairing simulations: using the same sequence of possible children for all children of a given node.
- Using the additional information from HCSP for ordering legal decisions in a given node; for example, we might:
  - consider moves in lexicographic orders on x-axis and y-axis;
  - consider moves in order of probability of mine, and in case of tie distance to the frontier (this is inspired by [11]);
  - consider random order (which avoids biases).
- Using progressive widening[2, 4, 10]; testing various coefficients, and maybe non-polynomial rules; or use rules depending on rewards obtained for the already sampled moves; see Fig. 3 for more on this.

Format	CSP-PGMS	HCSP	BSSUCT	OH
4 mines on 4x4	64.7 %	67.0%	70.0% ± 0.9%	67.0% ± 0.5%
1 mine on 1x3	100 %		100%	100%
3 mines on 2x5	22.6%	21.0%	25.4%± 1%	23.4% ± 0.5 %
10 mines on 5x5	8.20%	8.51%	9% (p-value: 0.14)	11.4% ± 0.4 %
5 mines on 1x10	12.93%	12.7%	18.9% ± 0.2%	17.0% ± 0.4 %
10 mines on 3x7	4.50%	4.76%	5.96% ± 0.16%	6.1% ± 0.2 %
15 mines on 5x5	0.63%	0.63%	0.9% ± 0.1%	1.15% ± 0.1 %
10 mines on 8x8		79.9 %		80.2 ± 0.48
10 mines on 9x9	80%	90.5%		89.9% ± 0.3%
40 mines on 16x16	45%	76.4% ± 0.4%		74.4% ± 0.5% (100 sims per move)
99 mines on 16x30	34%	38.1% ± 0.5%		38.7 ± 1.8 % (100 sims per move)

Table 1. Results of various implementations on the MineSweeper games. CSP-PGMS is the PGMS implementation of CSP (http://www.ccs.neu.edu/home/ramsdell/pgms/). HCSP is the implementation of CSP with heuristic breaking ties by playing as close as possible to the frontier between covered locations and uncovered locations[11]; results are averaged over 10^5 games except 16x30 which is averaged over 10^4 games. BSSUCT is the implementation of UCT from [13]. OH is our Optimistic Heuristics program. For OH, results are obtained with 10000 simulations per move, except expert mode and intermediate mode (99 mines on 16x30 and 40 mines on 16x16) which use 100 simulations per move. BSSUCT is not documented for cases in which it was too slow for being operational in [13].

- Simulate several times for evaluating a node, before starting new simulations from the root.
- Using rapid action value estimates[6] (either it brings an improvement, or it is an interesting counter-example; see Fig. 4 for a counter-example in the game of Go.).

More specifically for Minesweeper, we don't know if the assumption that the optimal move has minimum probability of mine is safe. We just break ties between various moves with minimum probability of mine, and we did not find any proof that doing otherwise (i.e. considering also moves with non minimal probability of mine) can bring an improvement; but we have no proof. Also we have no proof that always playing in the corner for the first move is a good idea; it is known as a good heuristic, but maybe in some cases there are better moves (with the rule "first move is a 0", it is mathematically proved that the center is a good move in some cases; see [3]).

Acknowledgements

The authors are grateful to NSC for funding NSC100-2811-E-024-001, to ANR for funding COSINUS program (project EXPLO-RA ANR-08-COSI-004), to the

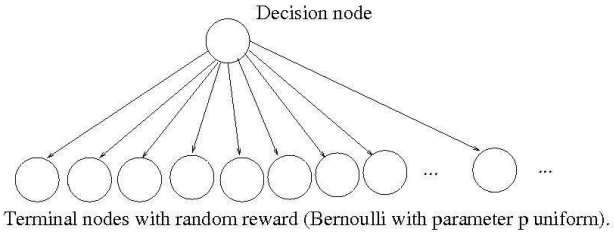

Fig. 3. Example in which progressive widening is necessary; each decision leads to a reward which is a Bernoulli random variable with parameter p, uniformly drawn in $[0, 1]$. There are infinitely many possible decisions. Without progressive widening, all moves are tested just once, and the final arm is chosen randomly, whereas a progressive widening as in [15] is mathematically consistent.

authors of PGMS and PGMS-CSP for making their implementations freely available. We are also grateful to the MASH European project (program FP7) and the the IOMCA (ANR) project.

References

1. M. M. Ben-Ari. Minesweeper as an np-complete problem. *SIGCSE Bull.*, 37:39–40, December 2005.
2. G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, and B. Bouzy. Progressive Strategies for Monte-Carlo Tree Search. In P. Wang et al., editors, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.
3. A. Couetoux, M. Milone, and O. Teytaud. Consistent belief state estimation, with application to mines. In *Proceedings of the TAAI 2011 conference*, page in press, 2011.
4. R. Coulom. Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.
5. S. Gelly. *Une contribution l'apprendissage par renforcement ; application au Computer GO.* PhD thesis, Université Paris-Sud, September 2007.
6. S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM Press.
7. K. B. Hein and R. Weiss. Minesweeper for sensor networks–making event detection in sensor networks dependable. In *Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 01*, CSE '09, pages 388–393, Washington, DC, USA, 2009. IEEE Computer Society.
8. R. Kaye. Minesweeper is np-complete. *Mathematical Intelligencer*, 22:915, 2000.
9. L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning (ECML)*, pages 282–293, 2006.
10. C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong. The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in games*, 2009.

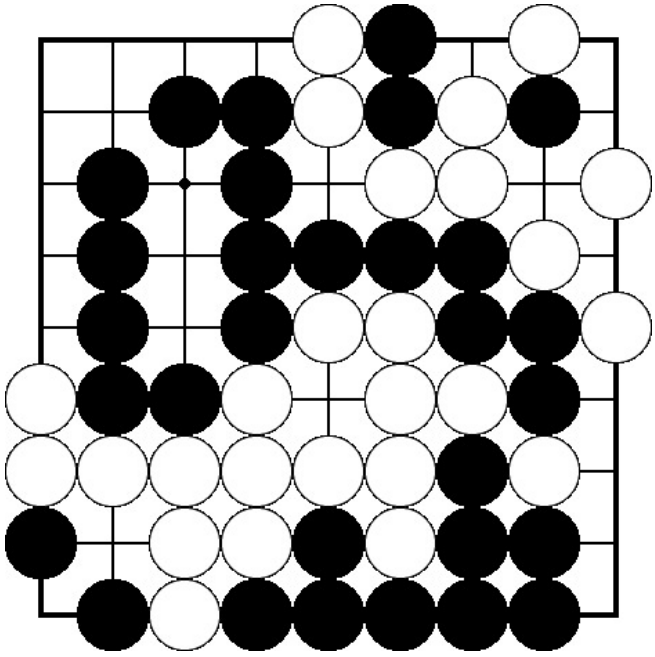

Fig. 4. Example provided by Martin Müller, in which RAVE values do not work well: WhiteB2 is the only good move (avoiding seki in bottom-left); but RAVE values are very bad for B2 because B2 only makes sense as a first move, and RAVE values propagate values from second time step, third time step, fourth time steps, ..., to the current time step.

11. M. Legendre, K. Hollard, O. Buffet, and A. Dutech. Minesweeper: Where to probe? Technical Report RR-8041, INRIA, 2012.
12. P. Nakov and Z. Wei. Minesweeper, #minesweeper, 2003.
13. M. Sebag and O. Teytaud. Combining Myopic Optimization and Tree Search: Application to MineSweeper. In *LION6, Learning and Intelligent Optimization*, pages in press (14 pages, long paper), Paris, France, 2012.
14. C. Studholme. Minesweeper as a constraint satisfaction problem. *Unpublished project report*, 2000.
15. Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, volume 21, 2008.